Active Monitoring of Dependency Models

The Invention

ADI (Active Dependency Integration) is an information and execution model for the description of enterprise systems, solutions, and services. It supports the modeling of various entities and business components (e.g. event, disk, application, activity, business process), the information that is associated with them (i.e. schema), and the semantic relationships among them (e.g. dependency between a business component and other business components and events). ADI execution model (based on AMIT-Active Middleware Technology) monitors and manages business components and the relationships among them. It automatically updates business components information in response to events occurrences and constraints violations, and propagates changes in business components to other business components according to the dependency model.

The Problem Solved

1. Modeling of a business (i.e. an enterprise system, a solution, or a service) and its behavior.
    1.1. Modeling of the business's components, the information associated with them and the events that affect or are signaled by the business's components.
    1.2. Modeling of the relationships between events and business components (i.e. the effect an event has on business comments).
    1.3. Modeling of the semantic relationships (i.e. dependencies) between business components and other business components (i.e. propagation of state changes in one business component to other business components).
2. Monitoring business behavior based on a model
    2.1. Monitor in runtime the effect of events on business components (e.g. when an event that reports on server utilization occurs, know its affect on the server).
    2.2. Monitor in runtime the effect of state changes in a business component on other business components (e.g. when the server fails, due to high utilization, know what is the effect of the failure on the applications that run on the server, on the activities that use these applications, and on the business itself)

Advantages of Using the Invention

1. Multiple domains: ADI can be deployed in any domain after minor configuration.
   1.1. Business components types are defined externally to ADI code.
   1.2. Dependency types can be defined easily using conditions on business component state, cardinality and ordering. The semantics of new dependency types are defined using AMIT rules externally to ADI code
   1.3. Examples for ADI applications domain are E-business management, Work-flow management, System management, and Web service management.
2. Expressive power:
   2.1. Ability to model all enterprise activities and components including objects, data, tasks, applications, resources, and business processes.
   2.2. Ability to define new dependency types and their semantics.
   2.3. Ability to enhance the semantics of defined dependency types using dependency composition.
   2.4. Ability to define rules that describe business behavior.
3. Efficient execution model (based on AMIT). Performance is linear in model size and independent in the number of events.
4. Integration with AMIT.
   4.1. AMIT situations can be used as events in ADI.
   4.2. ADI triggered events in response to change in a business components can be used in AMIT to compose situations.

ADI Rule Language

An XML language that enables the definition of

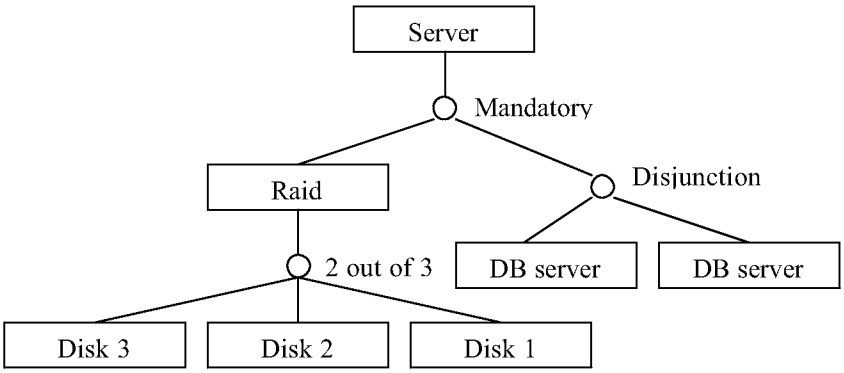
1. The business domain: event types, business component types, and dependency types.
2. Rules that describe how an event affects a business component
3. Rules that describe when a change in a business component triggers an event.
4. Description of business components and dependencies between them, including compound dependencies.
5. Rules that describe how information is propagated from one business component to another according to dependencies between them.

Efficient Execution Model

ADI receives event information and reports about changes in business components. When an event arrives to ADI

1. ADI finds the effect of the event on a business component, i.e., the change in a business component's state (e.g. a server becomes unavailable if the event server utilization reports on utilization that is higher than 90 percents).
2. If an event changes a business component state then ADI discovers how the change in one business component state propagates to other business components according to the business model.

The following model shows that a server requires both the raid and one of the database servers. The raid requires that two out of the three disks are available.



The dependency between the server and the database servers is a compound dependency. It is composed of the simple dependency types *mandatory* and *disjunction*.

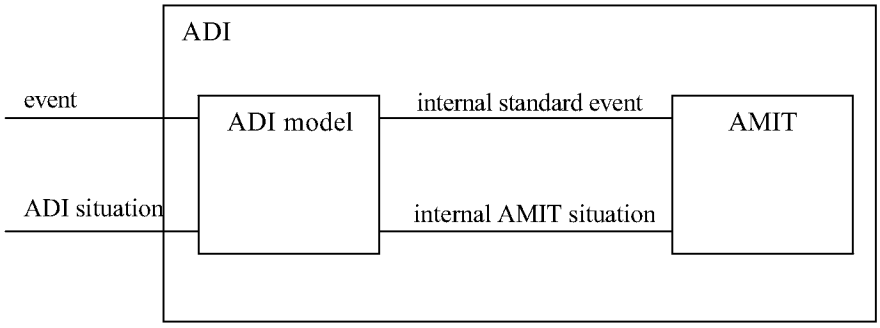
When an event that reports on disk1 failure occurs, the raid continues to function correctly and so does the server. However, when in addition an event that reports on disk2 failure occurs, the raid fails and as a result the server fails.

Dependency semantics is expressed using AMIT rules. In order to calculate state propagation ADI uses the predefined AMIT rules and the AMIT engine.

a. When a business component state changes, ADI creates an event that includes the following information and sends it to an internal AMIT engine:
   - The dependency id – for each dependency in the model ADI holds an internal id.
   - The place of the affected business component in the dependency – for each business component that participates in a dependency, ADI holds an internal id that aims at supporting dependencies that consider order of entities.
   - New state information

b. When a situation is detected by AMIT
   - ADI identifies the dependency that is reported by the situation
   - ADI identifies the target entities of this dependency
   - ADI updates state information of these entities. If state information changes then the previous step is repeated.
   - If required, ADI signals an event to the outside world to report about propagated state changes (ADI situation)

A compound dependency is treated as a virtual entity for the purpose of identifying state propagation (e.g. in the example above the mandatory dependency has two sources, a raid and a virtual entity representing the disjunction dependency; the disjunction dependency's target is a virtual entity representing the mandatory dependency).



AMIT situation describes the logic of a single simple dependency type
It is performed for every dependency according to the dependency id (key)
It is evaluated from the creating of the dependency until it is terminated (lifespan)
It considers state changes in any source business component in the dependency (operands)

Related Work in IBM

1. Gautam Kar/Watson/IBM

   Develops algorithms and processes that can be used to discover dependencies in an e-business environment; represents dependencies using standard modeling tools; and uses dependency information structure to identify Line of Business Views that relate business metrics to monitored resource metrics.

   Differences between Gautam's and ADI

   a. Gautam focuses on dependency discovery – ADI does not handle dependency discovery
   b. Gautam uses standard modeling tools – ADI enables the definition of new dependency types and their semantics using AMIT rules. Standard modeling tools define some predetermined dependency types (if any) without specific semantics.
   c. Gautam does not perform runtime monitoring of the effect of events on business components, and how state change in one business component affect other business components – ADI does.

2. Christine Draper/UK/IBM - xSM

   xSM is architecture for building management applications that manage solutions.

   Differences between xSM and ADI:

   a. xSM supports a predetermined set of dependency types - ADI enables the definition of new dependency types and their semantics using AMIT rules.
   b. xSM does not perform runtime monitoring on the model – ADI does.

3. Rosario Uceda-Sosa/Watson/IBM – IRIS

   IRIS is An Intelligent Information Infrastructure that provides dynamic views of data tailored to users needs and merges distributed data to form a coherent view.

   Differences between IRIS and ADI:

   a. IRIS supports a predetermined set of dependency types - ADI enables the definition of new dependency types and their semantics using AMIT rules.
   b. IRIS does not perform runtime monitoring on the model – ADI does.

Related Work not in IBM

1. Appilog PathFinder is a business service management solution that discovers IT assets, maps them to the business processes they support, and visualizes the effects of infrastructure events.
   Differences between Appilog PathFinder and ADI:
   a. PathFinder supports a predetermined set of dependency types - ADI enables the definition of new dependency types and their semantics using AMIT rules.
   b. PathFinder monitors the effects of infrastructure events – ADI can monitor the effect of events at all level, i.e., infrastructure events, applications events, business process events, etc.
2. Micromuse Netcool provides high level views of service available through its topology maps, allowing to manage the enterprise environment based on geography or other criteria.
   Differences between Micromuse Netcool and ADI:
   a. Netcool supports a predetermined set of relationships - ADI enables the definition of new dependency types and their semantics using AMIT rules.
   b. Netcool focuses on the business management domain – ADI is aimed for any domain.
   c. Netcool models are limited to the IT, applications and services – ADI can model entities in additional levels.
3. Managed Objects Formula unifies management data by dynamically integrating and consolidating it into a Web-enabled Object Integration Model.
   Differences between Managed Objects Formula and ADI:
   c. Formula supports a predetermined set of dependency types - ADI enables the definition of new dependency types and their semantics using AMIT rules.
   d. Formula monitors the effects of infrastructure events – ADI can monitor the effect of events at all level, i.e., infrastructure events, applications events, business process events, etc.

ADI in Products

ADI is part of the e-bMS service offering version 2.0 to be released in March 2003.

e-bMS follows the "autonomous computing" paradigm that IBM applies in different domains. It answers the following challenges:

1. Given a set of goals in the business process level (e.g., performance, diversified service level, availability), and a given configuration that describes the relationships between entities (business processes, activities, applications, resources), construct a set of monitoring rules that determine when a goal may be violated, and activate a correcting action where applicable
2. Ensuring the health and continuity of e-business systems

ADI provides a model-based view of the e-bMS managed enterprise in which the model definition derives the monitoring, management, and application development needs.